



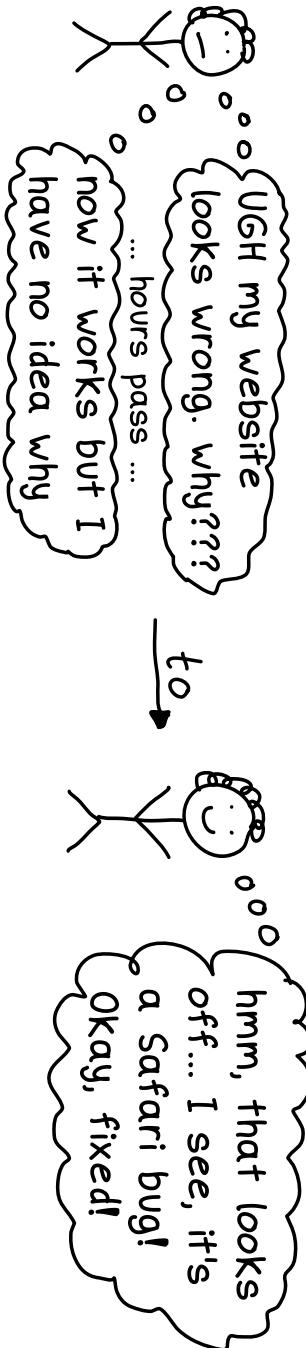
ISSUES YES MUCH



o this?
more at
★ wizardzines.com ★

What's this?

I wrote this zine because, after 15 years of being confused about CSS, I realized I was still missing a lot of basic CSS knowledge. Learning the facts in this zine helped me go from:



This zine also comes with 'examples' for you to try out. They're at:

<https://css-examples.wizardzines.com>

Panels which have examples you can try are labelled



thanks for reading

CSS is a HUGE topic and there's a lot more to learn than what's in this zine. Here are some of my favourite CSS resources:

Q CSS Tricks (css-tricks.com)

Hundreds of helpful blog posts and incredible guides, like their guides to centering & flexbox.

Q Can I use... (caniuse.com)

Tells you which browser versions (and what likely % of your users) have support for each CSS feature.

Q W3 (w3.org/TR/CSS)

The CSS specifications. Can be useful as a reference tool!

My favourite reference for CSS, JS, HTML, and HTTP

credits

Cover art: Vladimir Kašiković

Editing: Dolly Lanuza, Kamal Marhubi

Technical review: Melody Starling

and thanks to all the beta readers ☺

testing checklist

26

Finally, it's important to test your site with different browsers, screen sizes, and accessibility evaluation tools.

browsers	sizes	accessibility
<input type="checkbox"/> Chrome	<input type="checkbox"/> small phone (~300px wide)	<input type="checkbox"/> colour contrast
<input type="checkbox"/> Safari	<input type="checkbox"/> tablet (~700px)	<input type="checkbox"/> text size
<input type="checkbox"/> Firefox	<input type="checkbox"/> desktop (~1200px)	<input type="checkbox"/> keyboard navigation
<input type="checkbox"/> maybe others!		<input type="checkbox"/> works with a screen reader



performance

- fake a slow/high latency network connection!

table of contents

ATTITUDE

CSS isn't easy.....	4
CSS isn't design	5
CSS specifications.....	6
backwards compatibility.....	7
BASICS	
selectors.....	8
specificity.....	9
default stylesheets.....	10
units.....	11

AYOUT

inline vs block.....	12
the box model	13
padding & margin ..	14
borders.....	15
flexbox basics.....	16
CSS grid: areas!.....	17
centering.....	18
position: absolute.....	19

GETTING FANCY

hiding elements.....	20
stacking contexts.....	21
CSS variables.....	22
transitions.....	23
MAKING IT WORK	
media queries.....	24
the CSS inspector	25
testing checklist.....	26

CSS isn't easy

4

CSS seems simple at first

```
h2 {  
    font-size: 22px;  
}
```

Ok this is easy!

the spec can be surprising

setting overflow: hidden; on an inline-block element changes its vertical alignment

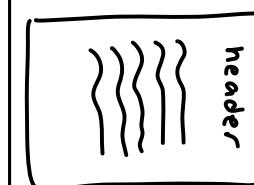
weird!

css 2.1

a layout like this is simple to implement!

TRY ME!

and it is easy for simple tasks



a layout like this is simple to implement!

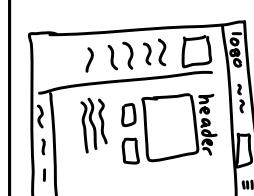
and all browsers have bugs

I don't support flexbox for <summary> elements

ok fine

safari

but website layout is not an easy problem



this needs to adjust to so many screen sizes!

accept that writing CSS is gonna take time

If I'm patient I can fix all the edge cases in my CSS and make my site look great everywhere!

the CSS inspector

25

all major browsers have a CSS inspector

usually you can get to it by right clicking on an element and then "inspect" element, but sometimes there are extra steps

see computed styles

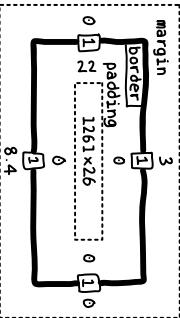
here's a website with 12000 lines of CSS, what font-size does this link have?

12px, because of x.css line 436

browser

see overridden properties

```
button {  
    display: inline-block;  
    color: var(--orange);  
}
```



look at margin & padding

▼ Box Model

edit CSS properties

```
element {  
    /* lets you change this element's properties */  
}  
  
button {  
    display: inline-block;  
    border: 1px solid black;  
}  
  
this lets you change the border of every <button>!
```

... and LOTS more

different browsers have different tools! For example, Firefox has special tools for debugging grid/flexbox

media queries

24

media queries let you use different CSS in different situations

```
@media (print){  
    media query  
    #footer {  
        display: none;  
    }  
    CSS to apply  
}
```

```
max-width & min-width  
@media (max-width: 500px) {  
    // CSS for small screens  
}  
@media (min-width: 950px) {  
    // CSS for large screens  
}
```

accessibility queries

you can sometimes find out a user's preferences with media queries

examples:

```
prefers-reduced-motion: reduce  
prefers-color-scheme: dark
```

you can combine media queries

it's very common to write something like this:

```
@media screen and  
(max-width: 1024px)
```

print and screen

screen is for computer/
mobile screens
print is used when
printing a webpage
there are more: tv, tty,
speech, braille, etc

the viewport meta tag

```
<meta name="viewport"  
content="width=device-width,  
initial-scale=1">
```

Your site will look bad on mobile if you don't add a tag like this to the <head> in your HTML. Look it up to learn more!

5

CSS != design

web design is really hard

wow, forms are way more complicated than I thought

writing CSS is also hard

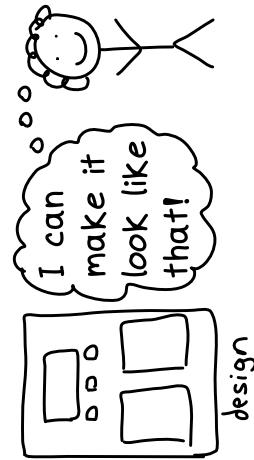
ok how exactly does flexbox work again?

remember that they're 2 different skills

hm, I have NO IDEA what I want this site to look like, maybe that's my problem and not CSS

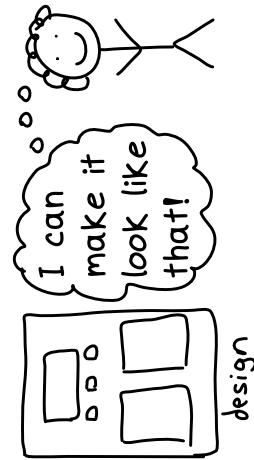
sketching a design in advance can help!

even a simple sketch can help you think!



title
D D D D
D D D D
D D D D

CSS is easier when you have a good design



I can make it look like that!
design

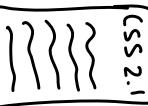
CSS

specifications

6

CSS has specifications

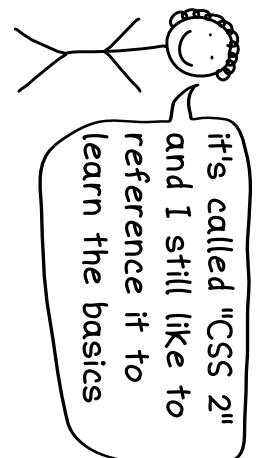
hello, this is how max-width works in excruciating detail



major browsers usually obey the spec

but sometimes they have bugs


oops, I didn't quite implement that right...



there used to be just one specification

today, every CSS feature has its own specification

you can find them all at <https://www.w3.org/TR/CSS/>

there are dozens of specs, for example: colors, flexbox, and transforms

levels

CSS versions are called "levels".

new levels only add new features. They don't change the behaviour of existing CSS code

new features take time to implement

 <https://caniuse.com> 

can tell you which browser versions support a CSS feature

transitions

23

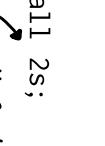
an element's computed style can change

2 ways this can happen:

- ① pseudo-classes (like :hover)
- ② Javascript code el.classList.add('x')

... unless you set the transition property

 TRY ME!

```
a {  
  color: blue;  
  transition: all 2s;  
}  
a:hover {  
  color: red;   
  will fade from blue to red over 2s  
}
```

transition has 3 parts

transition: color 1s ease;

which CSS duration timing function to animate

not all property changes can be animated....

list-style-type: square;

I don't know how to animate that, sorry! 

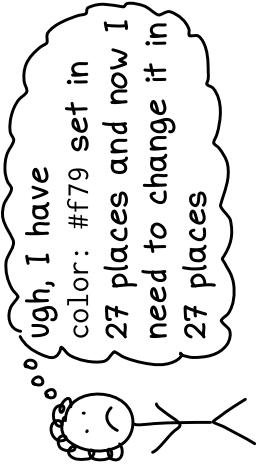
...but there are dozens of properties that can if it's a number or color, it can probably be animated!

```
font-size: 14px;  
rotate: 90deg;  
width: 20em;
```

CSS variables

22

duplication is annoying



define variables in any selector

```
body {  
    --text-color: #f79;  
}  
#header {  
    everything  
    --text-color: #c50;  
}  
#header  
    applies to children  
of #header
```

do math on them with calc()

```
#sidebar {  
    width: calc(  
        var(--my-var) + 1em  
    );  
}
```

use variables with var()

```
body {  
    color: var(--text-color);  
}  
variables always  
start with --
```

changes to variables apply immediately

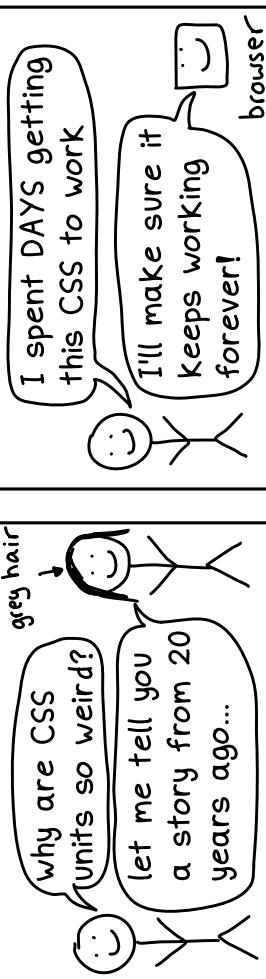
A character pointing at a speech bubble. The bubble contains text about CSS variable changes.

```
set --text-color  
to red  
ok everything  
using it is  
red now!  
JS  
CSS renderer
```

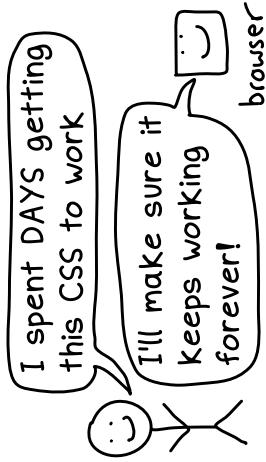
backwards compatibility

7

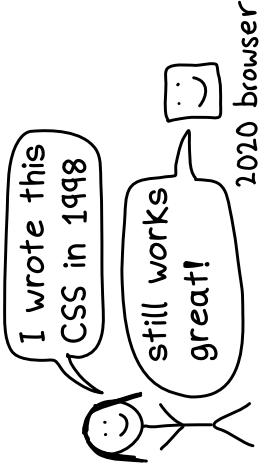
this makes CSS hard to write...



... but it means it's worth the investment

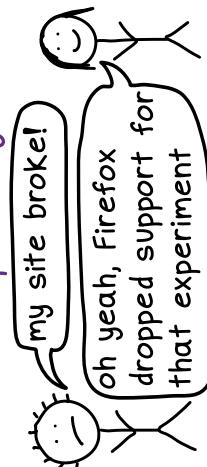


browsers support old HTML + CSS forever

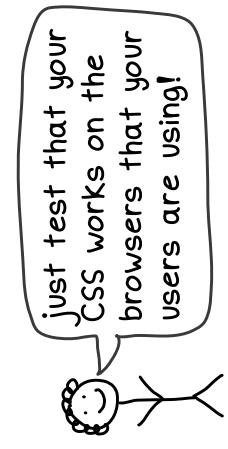


newer features are often easier to use

if you don't follow the standards, you're not guaranteed backwards compatibility



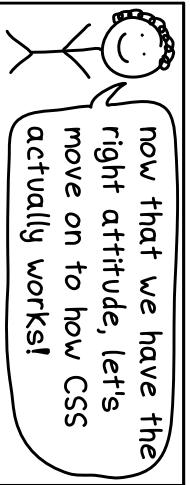
your CSS doesn't have to support browsers from 1998



what people expect from a website has changed a LOT since 1998. Newer CSS features make responsive design easy

a few CSS selectors

8



`div`

matches div elements

`<div>`

`#welcome`
matches elements by id

`<div id="welcome">`

`div.button`

match every .button element that's a descendant of a div

`<div class="button">`

`.button, #welcome`

matches both .button and #welcome elements

`.matches elements by class`

``

`div > .button`

match every .button element that's a direct child of a div

`<div> <button>`

`:checked`

matches if a checkbox or radio button is checked

`:checked`

`a:hover`

matches a elements that the cursor is hovering over

`tr:nth-child(odd)`

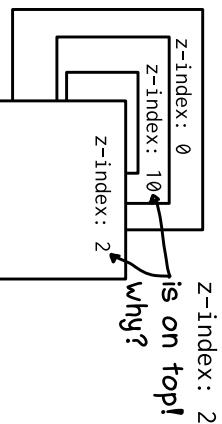
match alternating tr elements (make a striped table!)

Stacking contexts

21

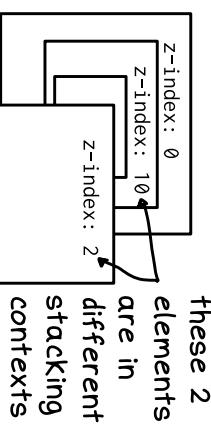
a z-index can push an element up/down...

```
.first {  
z-index: 3;  
}  
.second {  
z-index: 0;  
}
```

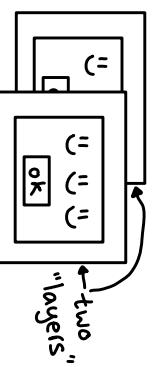


... but a higher z-index TRY ME! doesn't always put an element on top

every element is in a stacking context



a stacking context is like a Photoshop layer



by default, an element's children share its stacking context

setting z-index creates a stacking context

```
#modal {  
z-index: 5;  
position: absolute;  
}
```

this is a common way to create a stacking context

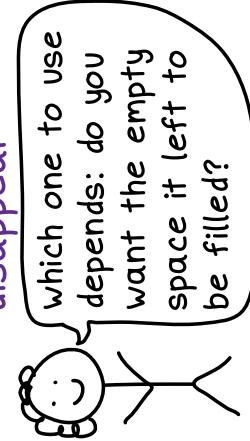
stacking contexts are confusing

You can do a lot without understanding them at all. But if z-index ever isn't working the way you expect, that's the day to learn about stacking contexts :)

hiding elements

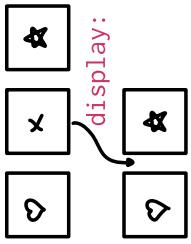
20

there are many ways
to make an element
disappear



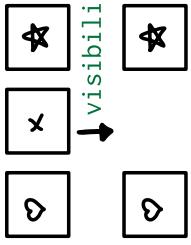
display: none;

other elements will move
to fill the empty space



visibility: hidden;

the empty space will
stay empty



opacity: 0;

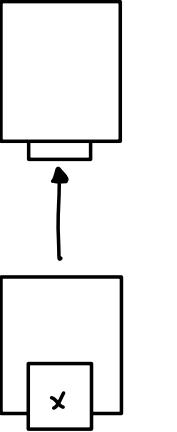
like visibility: hidden,
but you can still click on
the element & it'll still be
visible to screen readers.
Usually visibility: hidden
is better.

how to slowly fade out

```
#fade:hover {  
    transition: all 1s ease;  
    visibility: hidden;  
    opacity: 0;  
    set the opacity  
    just so that the  
    transition works
```

z-index

z-index sets the order
of overlapping positioned
elements



TRY ME!

specificity

Q

TRY
ME!

CSS uses the "most
specific" selector that
matches an element

In our example, the browser
will use color: orange because
IDs (like #start-link) are
more specific than
pseudoclasses (like :visited)

CSS can mix properties
from different rules

```
a:visited {  
    color: purple; ← if I use this  
    font-size: 1.2em; ← font-size  
}  
#start-link { ← but use this  
    color: orange; ← color because  
    #start-link is  
    more specific  
}
```

**different rules can set
the same property**

```
a:visited {  
    color: purple; ← which  
    font-size: 1.2em; ← one gets  
}  
#start-link { ← chosen?  
    color: orange;  
}
```

loses to →

how CSS picks the "most specific" rule

a selector with .classes
or :pseudoclasses
.sidebar .link {
 color: orange;
}

loses to →

a selector with an #id
#header a {
 color: purple;
}

loses to →

an !important rule*
color: blue !important;

loses to →

an inline style
style="color: green;"

loses to →

!important is very
hard to override,
which makes life
hard for your
future self!

default stylesheets

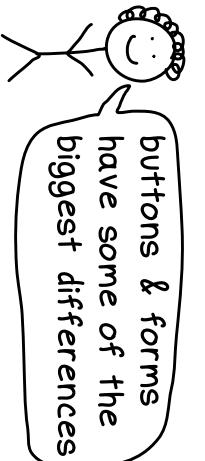
10

every browser has a default stylesheet
(aka "user agent stylesheet")

a small sample from the Firefox default stylesheet:

```
h1 {  
    font-size: 2em;  
    font-weight: bold;  
}
```

different browsers have different defaults



Firefox's default stylesheets are at:
<resource://gre-resources/>

every property also has a default "initial value"

the initial value (defined in the spec) is what's used if no stylesheet has set anything. For example, background-color's initial value is transparent

a CSS property can be set in 5 ways

- ① the initial value
- ② the browser's default stylesheet
- ③ the website's stylesheets
- ④ a user stylesheet (least common)
- ⑤ inline styles set with HTML/JS

lowest priority

highest priority

position: absolute

19

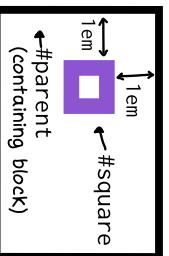
position: absolute; doesn't mean absolutely positioned on the page: it's relative to the "containing block"

the "containing block" is

the closest ancestor with a position that isn't set to static (the default value), or the body if there's no such ancestor.

Here's some typical CSS:

```
#square {  
    position: absolute;  
    top: 1em; left: 1em;  
}  
#parent {  
    position: relative;  
}  
this makes #parent the containing block
```



```
left: 0; right: 0; ≠ width: 100%;  
left: 0; right: 0;
```

```
width: 100%;
```

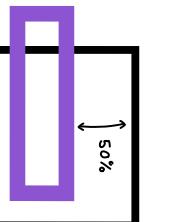


left and right borders are both 0px away from containing block

the box sticks out because width doesn't include borders by default

top, bottom, left, right will place an absolutely positioned element

```
top: 50%;  
bottom: 2em;  
right: 30px;  
left: -3em;
```



negative
works too

absolutely positioned elements are taken out of the normal flow

will a parent element expand to fit an absolutely positioned child?

nope!

centering

18

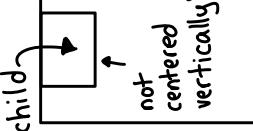
center text with
text-align

```
h2 {  
    text-align: center;  
}
```

center block elements
with margin: auto

```
example HTML:  
<div class="parent">  
  <div class="child">  
  </div>  
</div>
```

margin: auto
only centers horizontally



```
.child {  
    width: 400px;  
    margin: auto;  
}
```

vertical centering is easy with flexbox or grid

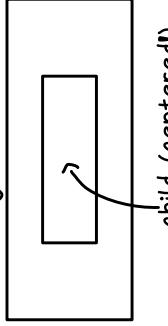
here's how with grid:

```
.parent {  
    display: grid;  
    place-items: center;  
}  
  
.child {  
    margin: auto;  
}
```

TRY ME!

it's ok to use a flexbox
or grid just to center
one thing

```
.parent {  
    display: grid;  
}  
  
.child {  
    margin: auto;  
}
```



units

CSS has 2 Kinds of units:
absolute & relative

absolute: px, pt, pc,
in, cm, mm

relative: em, rem,
vw, vh, %

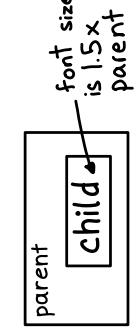
rem
the root element's
font size

1rem is the same
everywhere in the
document. rem is a
good unit for setting
font sizes!

TRY
ME!

em
the parent element's
font size

```
.parent {  
    font-size: 1.5em;  
}
```



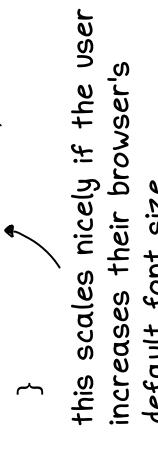
0 is the same
in all units

```
.btn {  
    margin: 0;  
}
```

also, 0 is different from none.
border: 0 sets the border width
and border: none sets the style

1 inch = 96 px
on a screen, 1 CSS "inch"
isn't really an inch, and
1 CSS "pixel" isn't really
a screen pixel.
look up "device pixel
ratio" for more.

```
.modal {  
    width: 20rem;  
}
```



this scales nicely if the user
increases their browser's
default font size

12 inline vs block

17

HTML elements default to **inline** or **block**

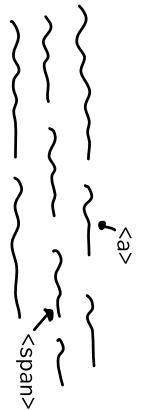
example inline elements:

```
<a> <span>
<p> <div>
<strong> <i>
<ol> <ul> <li>
<h1> - <h6>
<img> <q>
<code>
```

example block elements:

```
<div>
<pre>
```

inline elements are laid out horizontally



block elements are laid out vertically by default



inline elements ignore width & height*

Setting the width is impossible, but in some situations, you can use line-height to change the height

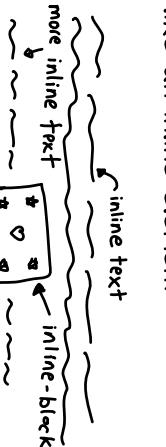
*ing is an exception to this: look up "replaced elements" for more

display can force an element to be inline or block

display determines 2 things:

- ① whether the element itself is inline, block, inline-block, etc
- ② how child elements are laid out (grid, flex, table, default, etc)

display: inline-block; inline-block makes a block element be laid out horizontally like an inline element



to get a different layout, use display: flex or display: grid

CSS grid: areas!

17

let's say you want to build a layout

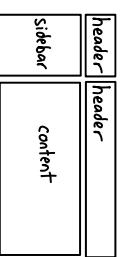


grid-template-areas lets you define your layout in an almost visual way

grid-template-areas:
"header header"

"sidebar content";

I think of it like this:



1. write your HTML



```
<div class="grid">
<div class="top"></div>
<div class="side"></div>
<div class="main"></div>
</div>
```

2. define the areas

```
.grid {
  display: grid;
  grid-template-columns:
    200px 800px;
  grid-template-areas:
    "header header"
    "sidebar content";
}
```

3. set grid-area

```
.top {grid-area: header}
.side {grid-area: sidebar}
.main {grid-area: content}
```

```
.top
.side
.main
```

flexbox basics

16

display: flex;

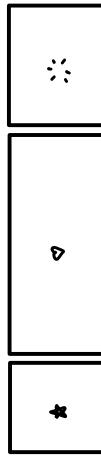
set on a parent element to lay out its children with a flexbox layout.

by default, it sets

flex-direction: row;

by default, children are laid out in a single row.
the other option is
flex-direction: column

flex-direction: row;



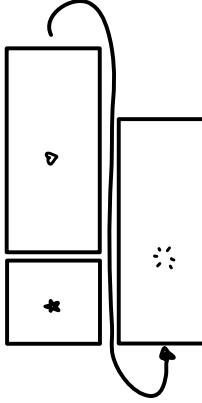
vertically center (or horizontally center if you've set
flex-direction: column)

justify-content: center;



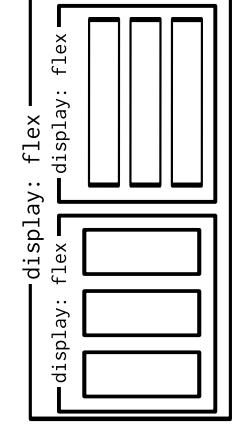
horizontally center
(or vertically if you've set
flex-direction: column)

flex-wrap: wrap;



will wrap instead of shrinking everything to fit on one line

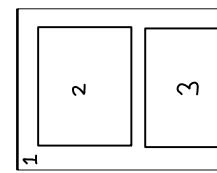
you can nest flexboxes



the box model

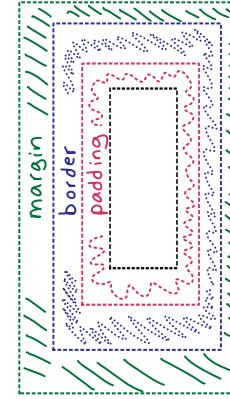
13

every HTML element is in a box

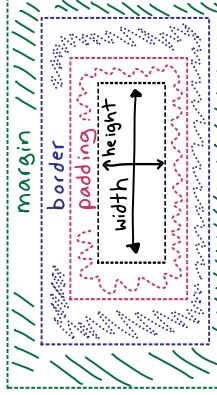


```
<div class="1">  
<div class="2" />  
<div class="3" />  
</div>
```

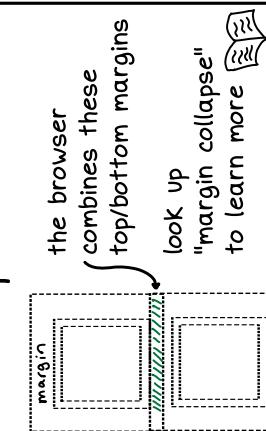
boxes have padding, borders, and a margin



width & height don't include any of those



margins are allowed to overlap sometimes



padding & border are inside the element, margin is outside

For example, clicking on an element's border/padding triggers its onclick event, but clicking on the margin doesn't.

padding + margin syntax

14

there are 4 ways
to set padding

all sides

padding: 1em;
vertical ↓ horizontal ↘
padding: 1em 2em;
top ↓ horizontal bottom ↗
padding: 1em 2em 3em;
top ↓ right ↗ bottom ↗ left ↙
padding: 1em 2em 3em 4em;

differences between padding & margin

→ padding is "inside" an element: the background color covers the padding, you can click padding to click an element, etc. Margin is "outside".

tricks to remember
the order

① trouble
top ↑ right ↗ bottom ↗ left ↙
② it's clockwise - 

you can also set padding on just 1 side

padding-top: 1em;
padding-right: 10px;
padding-bottom: 3em;
padding-left: 4em;

margin syntax is the same as padding

border-width also uses the same order:
top, right, bottom, left

borders

15

border has 3 components

border: 2px solid black;

is the same as

border-width: 2px;
border-style: solid;
border-color: black;

border-style options

solid

dotted

dashed

double

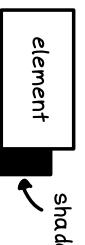
+ lots more
(inset, groove, etc)

border-{side} you can set each side's border separately:

border-bottom:
2px solid black;

outline

border-radius lets you have rounded corners
border-radius: 10px;
border-radius: 50%;
will make a square into a circle!

lets you add a shadow to any element

element shadow

box-shadow: 5px 5px 8px black;
x offset y offset blur radius
color

outline is like border, but it doesn't change an element's size when you add it

outlines on :hover/:active help with accessibility: with keyboard navigation, you need an outline to see what's focused